NAG Library Function Document

nag_double_quantiles (g01amc)

1 Purpose

nag_double_quantiles (g01amc) finds specified quantiles from a vector of unsorted data.

2 Specification

```
#include <nag.h>
#include <nagg01.h>
```

```
void nag_double_quantiles (Integer n, double rv[], Integer nq,
     const double q[], double qv[], NagError *fail)
```

3 Description

A quantile is a value which divides a frequency distribution such that there is a given proportion of data values below the quantile. For example, the median of a dataset is the 0.5 quantile because half the values are less than or equal to it; and the 0.25 quantile is the 25th percentile.

nag_double_quantiles (g01amc) uses a modified version of Singleton's 'median-of-three' Quicksort algorithm (Singleton (1969)) to determine specified quantiles of a vector of real values. The input vector is partially sorted, as far as is required to compute desired quantiles; for a single quantile, this is much faster than sorting the entire vector. Where necessary, linear interpolation is also carried out to return the values of quantiles which lie between two data points.

4 References

Singleton R C (1969) An efficient algorithm for sorting with minimal storage: Algorithm 347 *Comm. ACM* **12** 185–187

5 Arguments

1: **n** – Integer *Input*

On entry: the number of elements in the input vector **rv**.

Constraint: $\mathbf{n} > 0$.

2: **rv**[**n**] – double *Input/Output*

On entry: the vector whose quantiles are to be determined.

On exit: the order of the elements in **rv** is not, in general, preserved.

3: **nq** – Integer *Input*

On entry: the number of quantiles requested.

Constraint: $\mathbf{nq} > 0$.

4: **q**[**nq**] – const double *Input*

On entry: the quantiles to be calculated, in ascending order. Note that these must be between 0.0 and 1.0, with 0.0 returning the smallest element and 1.0 the largest.

Constraints:

$0.0 \leq \mathbf{q}[i-1] \leq 1.0$, for $i = 1, 2, \ldots, \mathbf{nq}$;
$\mathbf{q}[i-1] \leq \mathbf{q}[i]$, for $i = 1, 2, \ldots, \mathbf{nq} - 1$.

5: **qv**[**nq**] – double *Output*

On exit: **qv**$[i-1]$ contains the quantile specified by the value provided in **q**$[i-1]$, or an interpolated value if the quantile falls between two data values.

6: **fail** – NagError * *Input/Output*

The NAG error argument (see Section 3.6 in the Essential Introduction).

6 Error Indicators and Warnings

NE_BAD_PARAM

On entry, argument ⟨*value*⟩ had an illegal value.

NE_INT

On entry, **n** = ⟨*value*⟩.
Constraint: **n** > 0.

On entry, **nq** = ⟨*value*⟩.
Constraint: **nq** > 0.

NE_INTERNAL_ERROR

An internal error has occurred in this function. Check the function call and any array sizes. If the call is correct then please contact NAG for assistance.

NE_Q_NOT_ASCENDING

On entry, **q** was not in ascending order.

NE_Q_OUT_OF_RANGE

On entry, an element of **q** was less than 0.0 or greater than 1.0.

NE_STACK_OVERFLOW

Internal error. Please contact NAG.

7 Accuracy

Not applicable.

8 Parallelism and Performance

Not applicable.

9 Further Comments

The average time taken by nag_double_quantiles (g01amc) is approximately proportional to $\mathbf{n} \times (1 + \log(\mathbf{nq}))$. The worst case time is proportional to $\mathbf{n}^2$ but this is extremely unlikely to occur.

10 Example

This example computes a list of quantiles from an array of doubles and an array of point values.

10.1 Program Text

```
/* nag_double_quantiles (g01amc) Example Program.
 *
 * Copyright 2006 Numerical Algorithms Group.
 *
 * Mark 9, 2009.
 */

#include <nag.h>
#include <stdio.h>
#include <nag_stdlib.h>
#include <nag_stddef.h>
#include <nagg01.h>

int main(void)
{
  /* Scalars */
  Integer  exit_status = 0, i, n, nq;
  /* Arrays */
  double   *vec = 0, *quants = 0, *quant_vec = 0;
  /* Nag Types */
  NagError fail;

  INIT_FAIL(fail);

  /* Skip heading in data file */
  scanf("%*[^\n]");
  printf("nag_double_quantiles (g01amc) Example Program Results\n");
  scanf("%ld", &n);
  scanf("%ld", &nq);
  if (n >= 1 && nq >= 1)
    {
      if (!(vec = NAG_ALLOC(n, double)) ||
          !(quants = NAG_ALLOC(nq, double)) ||
          !(quant_vec = NAG_ALLOC(nq, double)))
        {
          printf("Allocation failure\n");
          exit_status = -1;
          goto END;
        }
    }
  else
    {
      if (n < 1)
        {
          printf("Invalid n.\n");
        }
      else
        {
          printf("Invalid nq.\n");
        }
      exit_status = 1;
      goto END;
    }
  for (i = 0; i < n; ++i)
    scanf("%lf", &vec[i]);
  for (i = 0; i < nq; ++i)
    scanf("%lf", &quants[i]);

  /* nag_double_quantiles (g01amc).
   * Find quantiles of set of values of data type double
   */
  nag_double_quantiles(n, vec, nq, quants, quant_vec, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_double_quantiles (g01amc).\n%s\n",
              fail.message);
      exit_status = 1;
      goto END;
    }
  printf("    Quantile   Result\n\n");
  for (i = 0; i < nq; ++i)
    {
      printf("   %7.4f     %7.4f\n", quants[i], quant_vec[i]);
    }

 END:
  NAG_FREE(vec);
  NAG_FREE(quants);
  NAG_FREE(quant_vec);

  return exit_status;
}
```

10.2 Program Data

```
nag_double_quantiles (g01amc) Example Program Data
22
5
0.5 0.729 0.861 0.44 0.791 0.001 0.062 0.912 0.27 0.141 0.32 0.133
0.654 0.285 0.553 0.438 0.316 0.696 0.718 0.293 0.704 0.029
0.0 0.25 0.73 0.9 1.0
```

10.3 Program Results

```
nag_double_quantiles (g01amc) Example Program Results
    Quantile   Result

    0.0000      0.0010
    0.2500      0.2737
    0.7300      0.6986
    0.9000      0.7848
    1.0000      0.9120
```